SKILL DESIGN MASTERCLASS

The Anatomy of a Perfect Skill

Reverse-Engineered from 100 Best Examples

6	100+	3-5×
Key Patterns	Skills Analyzed	Better Invocation

The 30-Second Refresher

A skill is a markdown file with YAML frontmatter. Claude auto-loads it when context matches — or you invoke it with /skill-name.

```
~/.claude/skills/

├── commit/SKILL.md        → /commit
├── code-review/SKILL.md   → /code-review
└── deploy/
    ├── SKILL.md           → /deploy
    └── scripts/helper.sh
```

YAML Frontmatter

Name & description at the top of every SKILL.md file

⚡ Auto-triggered

Claude scans descriptions and picks the best matching skill automatically

Manual invoke

Type /skill-name to force a specific skill anytime

PATTERN 1

Description Tells WHEN, Not Just WHAT

This is the single most important field. Skills with poor descriptions are invoked 3–5× less often.

✗ BAD

```
description: Code review tool
```

Only says WHAT — Claude doesn't know when to trigger this skill, so it rarely does.

✓ GOOD

```
description: Review code for bugs. Use when
reviewing PRs, checking quality, or user mentions
"review", "PR", "best practices"
```

Specifies WHAT + WHEN + keywords. Claude picks this reliably.

PATTERN 2

Be Directive, Not Conversational

Skills are instructions, not chat. Use imperative verbs. Claude follows directives far more reliably than polite questions.

WEAK (conversational)

Could you please review the code? Maybe check if there are any bugs?

- ❌ Vague
- ❌ No structure
- ❌ No output defined

STRONG (directive)

```
Review the current diff. Check for:
  1. Security vulnerabilities (OWASP Top 10)
  2. Performance issues (N+1 queries)
  3. Code style violations

Output: checklist with severity ratings.
```

PATTERN 3

Specify the Output Format

Most custom skills fail here. No format defined = inconsistent output every time.

WITHOUT output format

Generate a commit message for these changes.

Result varies every time:
→ Sometimes 1 line
→ Sometimes paragraphs
→ Sometimes with prefix
→ Inconsistent forever

WITH explicit format

```
type(scope): short description


- What changed

- Why it changed


type: feat|fix|refactor|docs

Max 50 chars, lowercase, present tense
```

PATTERN 4

Include the "Read First" Step

The best skills don't assume Claude knows your project — they tell Claude to look first.

1

Read the target file

Understand function signatures and types

2

Find existing tests

Read 1–2 examples from the test directory

3

Identify the framework

Jest, Vitest, Pytest, etc.

4

Note import patterns

Assertion style, import style to match

✓ **Result: Code that matches YOUR project's style — not generic code that breaks your linter.**

PATTERN 5

Define What the Skill Does NOT Do

Counterintuitive but powerful — 70% of high-quality skills include explicit out-of-scope sections.

WHY IT MATTERS

- → When a user asks for something out of scope, Claude doesn't try and fail
- → Triggers correct routing to a different skill
- → Reduces broken or unexpected outputs
- → Forces you (the skill author) to think about boundaries

From Anthropic's official PDF skill:

```
## Out of Scope


This skill does NOT:

- Handle scanned PDFs (use OCR skill)

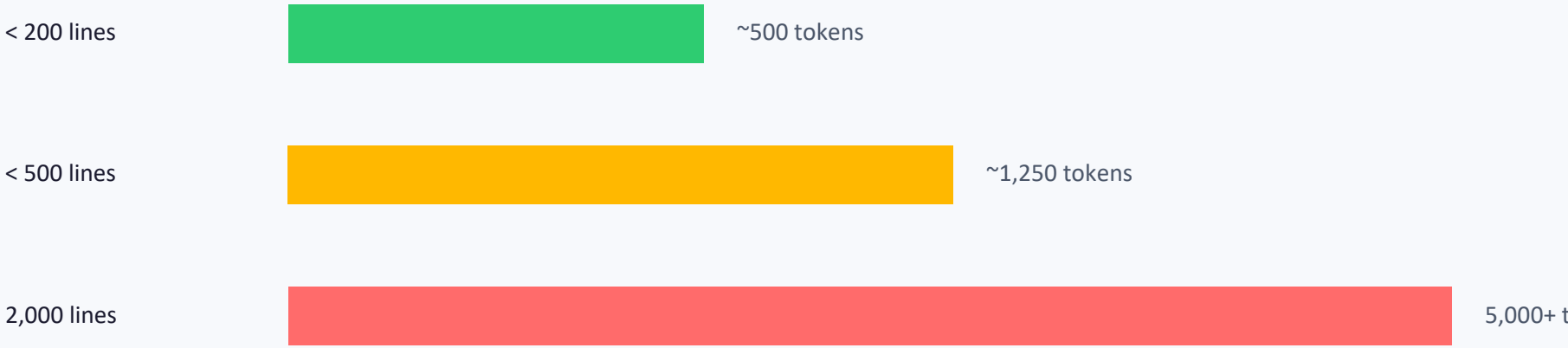
- Create PDFs from scratch

- Process password-protected files
```

PATTERN 6

Keep It Under 500 Lines

Every skill loads into Claude's context. Longer skills eat more tokens — and Claude loses focus halfway through.



Pro tip: Use Progressive Disclosure — SKILL.md under 200 lines + reference ADVANCED.md, EXAMPLES.md only when needed.

/commit — A Real-World Skill Using All 6 Patterns

```
---
name: commit
description: Create git commits from changes. Use when user says
"commit", "save changes" or after finishing a feature.
---

## Process
1. Run `git status` and `git diff`   ← READ FIRST
2. Group related changes logically
3. Generate commit: type(scope): description  ← FORMAT
4. Stage and commit each unit separately
5. Show summary: Created N commits: [list]

## Rules
Type: feat | fix | refactor | docs | test | chore ← DIRECTIVE

## Out of Scope  ← BOUNDARIES DEFINED
- Pushing to remote (use /push)
- Creating PRs (use /pr skill)
```

ALL 6 ✓

- ✓ WHEN in description
- ✓ Directive verbs
- ✓ Explicit format
- ✓ Read first step
- ✓ Out of scope
- ✓ < 50 lines

⚠ WHAT KILLS SKILLS

If your skill checks 3+ of these boxes, it's probably not firing or producing bad output.

DESCRIPTION FAILURES

- ✗ Under 50 characters
- ✗ No trigger keywords
- ✗ No use case context
- ✗ Inconsistent point of view

CONTENT FAILURES

- ✗ Conversational instead of directive
- ✗ No output format
- ✗ No 'read first' step
- ✗ No out-of-scope section
- ✗ Over 1,000 lines

DESIGN FAILURES

- ✗ Trying to do 5 things at once
- ✗ Hardcoded to one project
- ✗ Never updated after first write

THE 7-POINT FIX CHECKLIST

Pull up your weakest skill. Run through every item.

1	DESCRIPTION	Does description have ≥ 3 trigger keywords?
2	DESCRIPTION	Is the use case in the first 250 characters?
3	CONTENT	Are instructions directive — imperative verbs in numbered steps?
4	CONTENT	Is the output format specified explicitly?
5	CONTENT	Does it tell Claude to read existing files first?
6	SCOPE	Does it list what the skill does NOT do?
7	LENGTH	Is the SKILL.md under 500 lines?

Test: ask Claude something your skill should handle — without invoking it manually. If Claude picks the right skill ✓

THE COMPOUNDING EFFECT

Same Claude. Completely Different Output Quality.

✗ Eats context for nothing	✓ Loads fast, stays focused
✗ Inconsistent output every time	✓ Same reliable format, always
✗ Routes to wrong or no skill	✓ Perfect skill selection

6 months of well-written skills → a system, not a collection.